US-PAT-NO: 6499081

DOCUMENT-IDENTIFIER: US 6499081 B1

TITLE: Method and apparatus for determining a longest prefix match in a segmented content addressable memory device

DATE-ISSUED: December 24, 2002

US-CL-CURRENT: 711/108, 365/189.07 , 365/49 , 710/49 , 711/173

APPL-NO: 09/ 439834

DATE FILED: November 12, 1999

PARENT-CASE:

CROSS-REFERENCE TO RELATED APPLICATIONS

This application is a continuation-in-part of, and claims the benefit of, co-pending U.S. application Ser. No. 09/338,452 filed on Jun. 22, 1999 and entitled "METHOD AND APPARATUS FOR DETERMINING A LONGEST PREFIX MATCH IN A CONTENT ADDRESSABLE MEMORY DEVICE," which is a continuation-in-part of, and claims the benefit of, U.S. application Ser. No.:09/256,268 filed on Feb. 23, 1999 and entitled "METHOD AND APPARATUS FOR DETERMINING A LONGEST PREFIX MATCH IN A CONTENT ADDRESSABLE MEMORY DEVICE", now abandoned, Ser. No. 09/255,494 filed on Feb. 23, 1999 and entitled "METHOD AND APPARATUS FOR DETERMINING A LONGEST PREFIX MATCH IN A DEPTH CASCASED CONTENT ADDRESSABLE MEMORY SYSTEM", now abandoned, and Ser. No. 09/255,497 filed on Feb. 23, 1999 and entitled "TERNARY CONTENT ADDRESSABLE MEMORY CELL", now abandoned.

——— KWIC ———

Detailed Description Text - DETX (5):

For another embodiment, the CAM array is segmented into a number of CAM array blocks. Prefix logic circuits determine the longest prefix in each CAM array block from among CAM entries that match the search key. For one embodiment, the longest prefix for each CAM array block is then compared with the prefix mask data in that block and the match results provided to a priority encoder to determine the highest priority matching address. For another embodiment, the longest prefixes from each block are provided to compare circuitry that determines which of the longest prefixes is the longest for the entire CAM device. A block select circuit selects the CAM array block that stores the longest prefix for the entire CAM device for comparison with the prefix mask data in that CAM array block. The CAM index or address of the matching location may then be output from the CAM device. By restricting the search of the device-level longest prefix to only one CAM array block, power drawn by this comparison is reduced over searching all CAM array blocks in

parallel.

Detailed Description Text - DETX (105):

As described above, a single CAM device 300 of FIG. 3 can implement a search for a CIDR address that has the longest matching prefix. Additionally, multiple CAM devices 300 can be connected to form a depth cascaded CAM system and determine amongst themselves which of the CAM devices has the longest matching prefix for the entire CAM system. CAM device 300 can also have its CAM array 302 segmented or partitioned into a number of CAM array blocks each having a portion of the total number of available CAM entries. The prefix logic circuits determine the longest prefix in each CAM array block from among CAM entries that match the search key ("the block prefix"). The block prefix is then compared with the prefix mask data in its respective block and the match results provided to priority encoder 306 to determine the highest priority matching address in the entire CAM array.

Detailed Description Text - DETX (110):

Circuit 1902 provides to block select circuit 1904 an indication of which CAM array block stores the longest block prefix. In response, block select circuit 1904 provides block select signals BSEL to registers 316(1)-316(n). The block select signals select which register 316 outputs its block prefix to its respective CAM array block for comparison with the prefix mask data stored in the rows of local mask cells of that CAM array block. The results will be reflected on the mask match lines and encoded by priority encoder 306 to determine the CAM index or address of the matching location. Thus, the search between the longest block prefix and the prefix mask data is restricted to a single CAM array block. This can reduce power drawn or dissipated during this comparison operation to 1/n.sup.th of that when all CAM array blocks are searched simultaneously.

Detailed Description Text - DETX (126):

Block select circuit 2304 includes compare circuits 2306(1)-2306(4), priority encoder 2308 and decoder 2310. Each compare circuit 2306(x) compares a block prefix LPMBx with LPMD and generates a match signal MTCH(x) that indicates whether the block prefix matches the longest block prefix. Priority encoder 2308 and decoder 2310 translate the MTCH signals into the block select signals BSEL to select one of the CAM array blocks to perform the comparison of the longest block prefix with its prefix mask data. The logic state of the block select signals BSEL(1)-BSEL(4) may be qualified or timed by signal PFXTIM. For alternative embodiments, PFXTIM may be omitted. If each block prefix indicates a different number of contiguous higher-order prefix mask bits set to a no mask state, then only one match signal MTCH will be asserted, and the MTCH signals can be used directly as the block select signals BSEL. If, however, more than one block prefix matches LPMD, multiple MTCH signals are asserted. For another embodiment, priority encoder 2310 itself may generate the select signals BSEL.

Detailed Description Text - DETX (129):

FIG. 24 shows another embodiment of block select circuit 1904 and device LPM circuit 1902 of FIG. 19. For this embodiment, CAM array 302 is also segmented

or partitioned into four CAM blocks 302(1)-302(4). Any number of CAM array blocks can be used. Block select circuit 2404 includes block-bit compare circuitry 2402, priority encoder 2308 and decoder 2310. Block-bit compare circuitry 2402 compares one bit from each of the block prefixes LPMB1-LPMB4 with a corresponding bit from LPMD to generate signals MTCH(1)-MTCH(4). MTCH(1)-MTCH(4) indicate whether LPMB1-LPMB4, respectively, match LPMD. Priority encoder 2308 and decoder 2310 translate the MTCH signals into the block select signals BSEL to select one of the CAM array blocks to perform the comparison of the longest block prefix with its prefix mask data. For another embodiment, priority encoder 2310 itself may generate the select signals BSEL. The logic state of the block select signals BSEL(1)-BSEL(4) may be qualified or timed by signal PFXTIM. For alternative embodiments, PFXTIM may be omitted.

Detailed Description Text - DETX (134):

As described previously, the segmentation of CAM arrays can be further extended to include groups of CAM array blocks. FIG. 27 shows one example of extending the embodiment of FIG. 24 to include groups of CAM array blocks. As shown in FIG. 27, a left CAM array block group 2702(L) and a right CAM array block group 2702(R) provide block prefixes to left and right AND logic blocks 2302(L) and 2302(R), respectively. AND logic 2302(L) generates a longest block prefix LPMD(L) for the left group of CAM array blocks, and AND logic 2302(R) generates a longest block prefix LPMD(R) for the right group of CAM array blocks. Block select circuit 2704 includes block-bit compare circuit 2402(L) that generates MTCH(L) signals in response to LPMD(L) and block prefixes LPMB1-LPMB4. Block select circuit 2704 also includes block-bit compare circuit 2402(R) that generates MTCH(R) signals in response to LPMD(R) and block prefixes LPMB5-LPMB8. Priority encoder 2308 and decoder 2310 translate the MTCH signals into block select signals BSEL that indicate which block prefix is the longest block prefix. The block select signals BSEL also enable the longest block prefix to be provided to the appropriate CAM array block for comparison with the prefix mask data stored in that CAM array block.

Current US Original Classification - CCOR (1):

711/108

Current US Cross Reference Classification - CCXR (2):

365/49

Fast Routing Table Lookup Using CAMs

Anthony J. McAuley & Paul Francis[1]

Bellcore, 445 South Street, Morristown, NJ 07962-1910, USA

(mcauley@bellcore.com tsuchiya@bellcore.com)

Abstract

This paper investigates fast routing table lookup techniques, where the table is composed of hierarchical addresses such as those found in a national telephone network. The hierachical addresses provide important benefits in large networks; but existing fast routing table lookup technique, based on hardware such as Content Addressable Memory (CAM), work only with flat addresses. We present several fast routing table lookup solutions for hierarchical address based on binary- and ternary-CAMs and analyze the pros and cons of each.

1 Introduction

The central function of any communications switch is to route a call or packet to the appropriate destination. Simply put, this involves searching a routing table—a table composed of *<address, associated information>* entries—for the information needed to route the packet to the appropriate output port (or ports, in the case of multi-cast addresses). An entry in the routing table corresponding to a certain *address* tells the switch some *associated information* for deciding how to route the packet.

Ideally, routing table lookup should complete in the time it takes to read the packet off the link, or if cut-through switching is being performed (where the head of the packet is routed out before the tail arrives), the time it takes to read the address off the incoming link. As bandwidth and switching speeds increase, the time allotted to do the lookup decreases to the point where a software/RAM-based approach is not fast enough. A hardware structure called tries has been suggested [1]; but, for many lookup applications, it can be inefficient in memory or too slow. Exploiting the inherent parallelism of Content Addressable Memory (CAM) is attractive; but existing solutions [2] [3] [4] are limited to non-hierarchical addresses. This paper describes CAM-based architectures that provide low latency over a wide variety of address structures.

Sections 2 and 3 review the lookup function and address types. Section 4 categorizes existing lookup algorithms based on various memory hardware. Sections 5 and 6 describe six methods of using binary- and ternary-CAMs for routing table lookup. We conclude with a table for deciding between the solutions based on the address size and format.

2 The Lookup Function

This section describes the routing table lookup function that is executed every time a packet arrives at a switch.

2.1: Simplified Lookup Function

If the switch is connection-less (e.g. an IP network), then each packet header contains the complete addressing information needed to do the lookup function.

If the switch is connection-oriented (e.g. an ATM network), then most data packets need contain only a connection identifier. The complete addressing information (e.g. E.164 or IP address) is contained only in the first packet (e.g. SMDS) or in a special packet carried on the signalling channel (e.g. basic ATM). The connection identifier can be assigned by the sender or the switch; but, in either case, the connection identifier is a mnemonic representing all the address information.

The purpose of the connection identifier is increased efficiency. The connection identifier comes from a small number space compared to the address information. As a result, most of the packets of a connection do not need to carry all the addressing information, thus shrinking packet size. Further, the lookup function is simplified, because the size of the search field is reduced.

The connection identifier lookup function is a simplified case of the more general routing table lookup function. As such, any solution to the more general routing table lookup function, with its sparse address space, can be applied to the connection identifier lookup function (although it may not be the most cost-effective solution). We therefore focuses on the general routing table lookup function.

1. Recently published under the name of Paul Tsuchiya.

2.2: The General Routing Table Lookup Function

Up to this point, we have oversimplified by describing the input to the lookup function as being a simple address. However, the input can be both source and destination address, and other information that can collectively be called Quality-of-Service (QOS) information. QOS includes such *implicit* path information as low cost, low delay, high throughput, low error rate, and so on. This information is combined with the address for the lookup function. QOS information may also be *explicit*, such as an long-distance Inter-exchange Carrier (IEC) indication. In this case, the QOS information may take priority over the addressing information altogether, for instance because a switch is only concerned about getting the packet to the appropriate IEC. This priority effect also exists with hierarchically structured addresses. Here, one part of an address takes priority over another part in the routing decision.

For the remainder of this paper, we consider an *address* to potentially include both source and destination address and the QOS information described above. The way the addresses are assigned in a network affect the routing table lookup function; therefore we next look at three general address structures. This covers the full range of address types that one is likely to see in practice. In particular, it covers a wider range than previous work on hardware-assisted routing table lookup [1].

3 Three Types of Address

This section considers two addressing structures (flat and hierarchical) and two hierarchical address assignment algorithms (fixed-position and variable-position).

3.1: Flat Addresses

The simplest addressing structure is a flat address space, where we simply assign each destination a unique address chosen anywhere from the address space. This is seen, for example, in ethernet addresses and for local connection identifiers. This method has the advantage of simplicity; but is limited to small networks, where routing table size is manageable.

3.2: Fixed-position Hierarchical Addresses

Hierarchical addresses, such as those in the telephone system, greatly reduce routing table size. Figure 1 shows a small example, representing a subset of seven nodes (ovals) visible to a switch (not shown) with three levels of hierarchy. The telephone numbers are the standard US 10-digit code: where the "X" digit is a wild card, meaning that any digit matches that position. Table 1 shows the corresponding routing table. This routing table has addresses with four different fields: one 10-digit, two 6-digit, three 3-digit, and one 0-digit. For example, the switch has direct connections to two 6-digit switches that handle the 201-829 (via Port B) and 201-876 (via Port C) exchanges. The final entry is a 0-digit default switch that routes to everything else (via Port G).

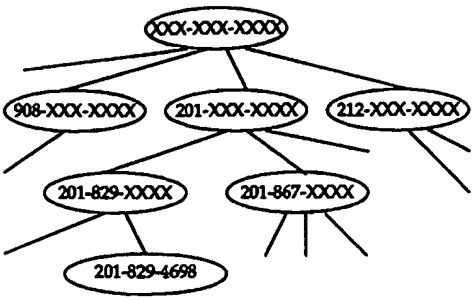

Figure 1 Fixed-position address hierarchy (with decimal addresses)

Table 1
Hierarchical address example

Address (decimal)	Next Hop
201-829-4698	Port A
201-829-XXXX	Port B
201-876-XXXX	Port C
201-XXX-XXXX	Port D
212-XXX-XXXX	Port E
908-XXX-XXXX	Port F

Consider a packet with address 201-829-4484. It matches three of the entries (201-829-XXXX, 201-XXX-XXXX, and XXX-XXX-XXXX). In this case, the best match is the one that matches on the most non-wildcard digits; thus the 201-829-XXXX represents the best route (port B), and perhaps the only correct route. A packet with address 908-829-4698 would be forwarded over Port F. Here, even though the 829-4698 part of the address matches 7 digits in the first entry, *any* mis-matched digits (908 instead of 201) results in a mis-match.

We can model the general lookup function as follows. The routing table consists of a list of *address/mask* pairs, and the associated information that gets returned as a result of the lookup. For example, Table 2 shows how the entries in Table 1 would be stored. The input to the lookup function is the *packetAddress*. If the *size* of a mask is the number of 1 bits in the mask, the result of the lookup function is the associated information of the entry with the largest mask such that: mask&packetAddress = address, where & is the

Table 2
Address and mask for Table 1

Address (decimal)	Mask (hex.)
201-829-4698	FFF-FFF-FFFF
201-829-0000	FFF-FFF-0000
201-876-0000	FFF-FFF-0000
201-000-0000	FFF-000-0000
212-000-0000	FFF-000-0000
908-000-0000	FFF-000-0000
000-000-0000	000-000-0000

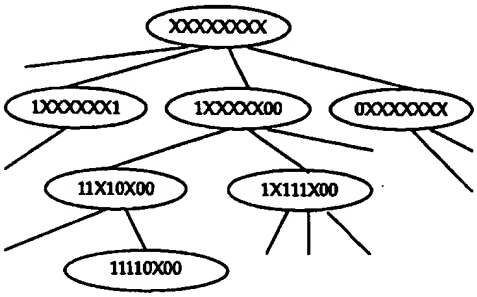

Figure 2 Variable-position address hierarchy (with binary addresses)

bitwise logical AND function. The flat routing table lookup, then, is a special case of the hierachical lookup function where the masks are all 1's.

Unlike the above example, just because two addresses are at some level *i* of the addressing hierarchy does not mean that they have the same mask (as it does with for instance telephone addresses in the USA). For instance, subnet numbers in IP addresses can fall on arbitrary bit boundaries [5]. Notice also that the 1's in the mask need not be contiguous. Indeed, there are known address assignment algorithms that can efficiently utilize the address space by taking advantage of variable-position (and in some cases non-contiguous) masks. Examples of this are kampai addressing [6], and variable-position subnet number assignment [7].

3.3: Variable-position Hierarchical Addresses

Figure 2 shows a view of a network similar to that used in the example of Figure 1, with seven areas (ovals) and three levels of hierarchy, but employing variable-position addresses (numbers inside the ovals). An area has the same locality for routing as the example of Figure 1.

Table 3
Variable-position addressing example

Address (binary)	Next Hop
11110X00	Port A
11X10X00	Port B
1X111X00	Port C
1XXXXX00	Port D
0XXXXXXX	Port E
1XXXXXX1	Port F
XXXXXXXX	Port G

Table 4
Address and mask for Table 3

Address (binary)	Mask (binary)
11110000	11111011
11010000	11011011
10111000	10111011
10000000	10000011
00000000	10000000
10000001	10000001
00000000	00000000

The numbers inside the ovals represent the addresses that switch has control over: that is, the numbers which are below it in the hierarchy. For example, the bottom oval in Figure 2 (11110X00) has two addresses associated with it: 11110000 and 11110100. The oval above this one (11X10X00) has four addresses associated with it: 11010000, 11010100, 11110000, and 1111100. From this small example, we can see that the addresses are much less rigidly structured than in the telephone example. Each area has a number of addresses equal to some power of 2.

If a switch has access to the seven areas shown in Figure 2, it will have a variable-position addressing table like that shown in Table 3. Table 4 shows the address and mask derived from Table 3. The lookup operation is the same as that already described for fixed-position hierarchical addresses (matching entry with largest mask).

For the routing table lookup function, there are important differences between fixed-position hierarchical addressing and variable-position hierachical addressing. The main difference with respect to this paper is that there are potentially many more mask combinations that must be considered in the lookup function for variable-position addressing. The consequences of this will be brought out

later in the paper. (The other major difference is that variable-position masks are not necessarily contiguous. However, this difference only affects software-driven routing table lookup, and so is not a factor in this paper.)

4 Routing Table Lookup

This section briefly reviews the general approaches to the lookup, using RAM, binary-CAM and ternary-CAM.

4.1: RAM-based Lookup

The RAM has two major operations:

- Write an entry into a specific address
- Read an entry by its address.

There are a number of (software) algorithms used to perform the lookup function using standard Random Access Memory (RAM).

A RAM can be perform the lookup in a single cycle if the data being searched (i.e. the packet address) is used as a direct index (RAM address) into memory. In this case, the size of a RAM is determined by the size of the search field. For example, with a 16-bit search field the RAM size is 64K (2^{16}) words. The number of words stored in a RAM has no effect on this size and cost. Thus, if there were only 256 words, each with a 16-bit search field, the RAM must still have 64K words. The size and cost of the RAM when used as a direct index grows exponentially with the search field. With current RAM technology trends, a 24 bit search pattern is the practical limit of an economic RAM-based search engine.

A linear search is the most efficient algorithm for table lookup, requiring only one entry per active address. If the entries in the routing table are searched in order of largest mask first, then the first match will be the best match. Of course, the linear search runs in time $O(N)$, where N is the number of entries, and so can take considerable time.

A faster approach is to form a tree search: using a binary tree, a patricia tree, a trie tree, and so on [8]. In general, these trees can push the search time towards $\log N$. In the case of the binary and patricia trees, the log base is 2. The log base can be increased, thus reducing search time, but for sparsely populated address spaces, this can result in unacceptable memory requirements [1]. Furthermore, even this search time may be excessive. For a high speed switch, the allotted search time can be measured in a few tens of instructions or less.

Under good conditions, a hash function can execute the lookup function in constant time [8], only slightly slower than direct access. The worst case search time, however, can be considerably worse than that of the tree searches. The performance is a function of the size of the hash memory and the number of addresses that must be searched in a given time window (after which a hashed entry will be timed out). While the routing table, because of hierarchical addresses, might be relatively small (10's or 100's of entries), the number of addresses that might potentially be searched can be large. This number depends on user traffic patterns, that can be hard to predict, especially for emerging data networks. Therefore, the amount of memory or the search time needed for hashing might be unacceptably large.

4.2: Binary-CAM-based Lookup

A CAM has three major operations:

- Write into the next free location
- Search for a word match.
- Read matching entries.

Data may be transferred to or from an CAM without knowing the memory address[2] of the word [4]. Binary data is automatically written to the next free word. To read a word the user must first do a search operation. Then, if there are multiple matches, the CAM decides (based on some internal state) which matched word to read next. Reading is useful because a CAM word has two parts. The most important part is the *search-field*, which is the part of the word that is matched with the search pattern. This typically contains the addresses of the known destinations. The CAM word also contains a *return-field*, which is the information returned during a read. This contains either the related information or an index.

All the CAM lookup algorithms are similar to the parallel search used in a RAM; however, the size of a CAM needed for direct access is determined primarily by the number of words that require storage. The size of the search field only affects the number of bits a word requires. For example, with a 10-bit search field, 10 bits per word are required. Thus, if there were only 256 possible inputs, each with a 16-bit search field, the CAM must have 256 words - with each word being at least 16 bits. The size and cost of a CAM grows linearly with the size of the search field and the number of entries.

The simplest CAM application is filtering, because it does not require any returned information other than the existence of the address. For example, to implement the network address screening function in SMDS, a CAM can be loaded with a list of valid/invalid network addresses. When a packet arrives, its address is used to search the CAM. Only if the CAM flag indicates a match/no-match is the packet processed.

If the amount of associated information is small (e.g. an output port index number), the CAM word is can store the

2. Some CAMs require addresses [3]; but addressless-CAMs are preferred for networking applications, because they directly store related information and reduce overall complexity [4].

related information in full. This direct access is fast (requires a single CAM read) and has low complexity. However, because the size of the CAM word is limited (by cost), the associated information must be relatively small, currently the maximum economic size is around 100 bits.

If the amount of associated information is large (e.g. because a large lower layer encapsulation address is required, or because multiple outputs are listed in the case of multicast forwarding), the CAM word is unable to store the related information. It therefore stores a unique index. The index is read, just as with direct access; but now the index is used as an address to read a RAM. This indirect access requires both a CAM read and a RAM read. Indirect access is therefore slightly slower and more complex than direct access; but allows more associated information.

4.3: Ternary-CAM-based Lookup

A ternary-CAM [9] has the same three major operations as a binary-CAM; but, while a binary-CAM stores one of two states (0 and 1) in each memory location (i.e. in each bit of a word), a ternary-CAM stores one of three states in each memory location (and also allows the search pattern to be one of the three states). We represent the three states by: 0, 1, and X. A ternary-CAM stores a don't care condition in the extra state (X), effectively allowing each word its own personal mask register. For hierarchical addresses, the ternary-CAM allows the address and mask information to be combined in a single ternary word.

We will introduce algorithms to perform lookup using a ternary-CAM in Section 6.

4.4: Cost Comparison

As a first order approximation, we assume the cost per bit of a:

- Binary-CAM is ten times the cost of a RAM.
- Ternary-CAM is twice the cost of a binary-CAM.

Since the entries in general routing tables tend to be sparsely populated over the (network) address space, direct indexing of the packetAddress into RAM is prohibitive. Therefore, it is necessary to either use a slower, software-driven search of RAM, or go to a hardware-based approach such as CAM (or a hybrid). Often, the time of the software-driven RAM search is unacceptable. With higher speeds, at some point it is necessary to go with the faster and well-bounded search time of a CAM.

Thus, although approximately an order of magnitude more expensive in terms of hardware, CAM-lookup solutions can offer superior performance compared to even the most sophisticated RAM-based search algorithms (careful management of the scarce CAM resources can help reduce costs - see Appendix A).

5 Binary-CAM Lookup Algorithms

This section describes three binary-CAM lookup algorithms/architectures: B1, B2, and B3 (B1 and B2 are well known and are included only for completeness).

5.1: B1 (Single-Cycle Single-Logical CAM)

A binary-CAM is directly useful for flat address lookup (or lookup for a switch at a fixed position in the hierarchy).

The system loads the CAM words with the routing table: i.e. the address and the associated information or index. The system also loads the mask register so that only the address is matched during a search (the associated information is masked). After loading the address, associated information, and mask, the CAM is ready to perform routing table lookup.

When a packet arrives at the switch, the system extracts its packetAddress. This packetAddress is used to ***search*** the CAM. If there is a match, the system ***reads*** the associated information in the subsequent cycle. After initialization, the CAM returns the associated information or index in two cycles: *search-read* (CAM search and CAM read).

If a new entry needs to be loaded into the routing table, the system adds the entry into the next free location. If an old entry needs to be removed from the table, the system selectively deletes that entry.

Method B1 is fast and has low complexity; but is only suited to flat addressing applications, because it only has a single fixed mask and assumes there is only one match.

5.2: B2 (Multiple-Cycle Single-Logical CAM)

If it is necessary to use more than one mask register, such as in hierarchical addressing, we cannot use method B1. We must be able to use different masks and have multiple search operations. A binary-CAM using multiple cycles works well for fixed-position hierarchical address lookup.

The system loads the CAM words with the address and the associated information (or index); but does not fix the mask register.

When a packet arrives, the system extracts its packetAddress and begins multiple mask loading and search operations. In the first cycle, the system loads the mask register of the address furthest down in the hierarchy (see Figure 1): that is, with the largest (most 1's) mask in the routing table (see Table 2). Then it searches (using the packetAddress) the CAM for a match. If a match occurs, it reads the associated information or index. If no match occurs, the second cycle begins and the system loads the mask register with the mask containing the second most 1's. The system again searches for a match. If a match occurs, it reads the associated information or index. If no match occurs, a third cycle begins. The system continues

through all the masks until a match is found (or if all else fails there is a default—a mask with all 0's).

In each cycle at most one match can occur. This is because masks only match things in the hierarchy at the level of the mask and below. Since the most specific (lowest in the hierarchy) mask is tried first, and since the routing table lookup sequence ends with the first match, we can have only one match. For fixed-position hierarchical addresses, the worst case number of cycles is twice the number of hierarchy levels plus one: *write-search-write-search-.....-read* (Mask write, CAM search, Mask write, CAM search,....CAM Read).

If a new entry needs to be loaded into the routing table, the system adds the entry into the next free location. If an old entry needs to be removed from the table, the system selectively deletes that entry.

Method B2 is slower and more complex than method B1; but is better with fixed-position hierarchical addresses. The number of cycles is bounded by the number of different masks, which for fixed-position hierarchical addresses is equal to the depth of the address hierarchy. (Note that the number of masks with variable-position hierarchical addresses can be much larger than the hierarchy depth, so this method may not be good for such addresses.) Since the address hierarchy tends to be shallow (the international telephone network address is only four levels), the number of cycles is small[3].

Unfortunately, the system must keep track of where it is, load the mask register with a different value each cycle, and check for completion. A faster, simpler alternative, if suitable hardware is available, is method B3.

5.3: B3 (Single-Cycle Multiple-Logical CAM)

In a single logical CAM, words share the mask register and data bus; so, if only a single search cycle is allowed, all words must receive the same search pattern. (A single logical CAM may be composed of more than one chip; but the same mask is applied uniformly to all words). A single logical binary-CAM cannot provide hierarchical addressing with a single search-read cycle. This section describes the use of multiple logical CAMs for hierarchical address lookup with a single search-read cycle. This is particularly useful for fixed-position hierarchical addressing, such as exists in the current telephone numbering scheme.

When addresses are hierarchical, two problems prevent table lookup in one cycle with a single logical CAM:

3. The number of cycles will on average be less than half the hierarchy depth, because method B2 begins with the larger masks - and larger masks are for frequently used: 1) nearby destinations that are low in the hierarchy, or 2) high access links that are higher in the hierarchy, but uses more detailed information.

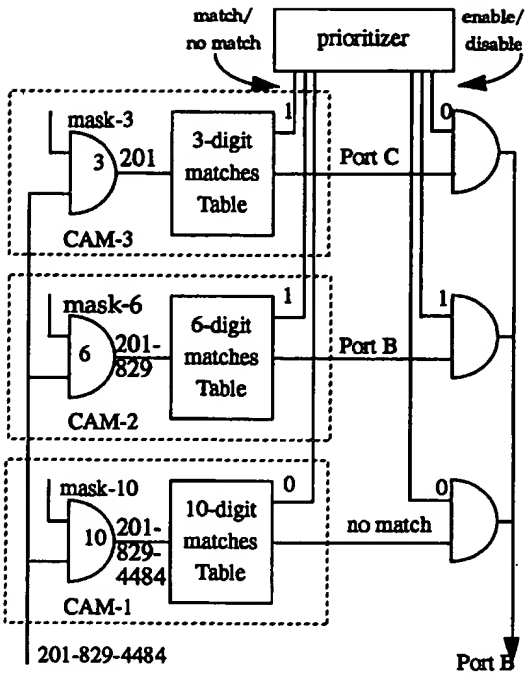

Figure 3 Lookup with multiple logical CAMs

- *variable "field" matching* (different addresses need different masks),
- *"best" matching*, (choose the winning match chosen from the multiple entries that can match a given address).

To overcome the variable field matching problem we use multiple *logical CAMs*. Words in different logical CAMs have their own mask registers. The system chooses which logical CAM to store data and sets its mask value. Therefore, even though they receive the same search word, each logical CAM has its own search pattern, one for each different mask.

To overcome the simultaneous matching problem we add logic at the output to filter through only the best match. This *match prioritizer* automatically selects the best match.

Figure 3 shows a system with three logical CAMs and match prioritizer, that translates the hierarchical addresses shown in Figure 1.

The system stores the addresses in different CAMs depending on their depth in the hierarchy. It stores all ten digit addresses (e.g. 201-829-4698) in the bottom logical CAM, and sets its mask (mask-10) to FFF-FFF-FFFF (see Table 2). It stores the six digit addresses in the middle logical CAM, and sets its mask (mask-6) to FFF-FFF-0000.

Finally, it stores all the three digit addressees in the top logical CAM, and sets its mask (mask-3) to FFF-000-0000. The final entry in table I does not need a CAM, because it is simply the default if no matches occur. With each logical CAM masking out the bits that are its wild-cards, the CAMs generate the correct matches in one search cycle.

When a packet arrives at the switch, the system extracts its packetAddress. This packetAddress is used to *search* all the logical CAMs simultaneously. If there are multiple matches, the match prioritizer distinguishes which is the best match. For the example shown in Figure 3, where the packetAddress is 201-829-4484, the top two CAMs produce matches. Each logical CAM has a fixed priority and a single output is guaranteed. For the example shown in Figure 3, the prioritizer picks the lowest logical CAM which produces a match: in this case it picks the middle CAM. The prioritizer only enables one buffer (the middle buffer in this example) to drive its signal (indicating port B in this example) onto the output bus. After initialization, the CAMs return the associated information or index in two cycles: *search-read* (CAM search and CAM read).

If a new entry needs to be loaded into the routing table, the system adds the entry into the next free location of the corresponding logical CAM (which stores entries with the same number of active digits). If an old entry must be removed from the table, the system selectively deletes that entry from the corresponding logical CAM.

Method B3 is fast and has relatively low complexity. It adds extra hardware; but this can be economically integrated onto a single chip. Also, it allows only a limited number of masks; but, as we have said, a few masks are sufficient for fixed-position hierarchical addresses. As the number of masks increase, however, it becomes more economical to use a ternary-CAM.

6 Ternary-CAM Lookup Algorithms

The section describes how the ternary-CAM's built in mask registers can be used to overcome the variable field matching problem for hierarchical addresses (in essence, a ternary-CAM is a multiple logical CAM system, where every word is a separate logical CAM).

6.1: T1 (Single-Cycle Single-Logical CAM)

Method T1 exploits the CAMs predictable ordering when reading multiple matched data. For example, the RCAM [4] always reads the matching entry that was entered first (in other words, the RCAM is *First In First Match* (FIFM)). If the system empties the CAM completely, then it knows that the order in which it puts the addresses into the CAM will also be the priority order.

The system stores the addresses (and their associated information or index) in order, starting with the lowest addresses (largest mask) in the hierarchy. Since we know that multiple matches will be read out lowest level addresses first, the "best" match is guaranteed. Table 5,

Table 6
Variable-position Address Priority

Address (ternary)	Implicit priority
11110XX0	7
11X10X00	6
1X111X00	5
1XXXXX00	4
1XXXXXXX	3
1XXXXXX1	2
XXXXXXXX	1

Table 5
Fixed-position Address Priority

Address (decimal)	Implicit priority
201-829-4698	7
201-829-XXXX	6
201-876-XXXX	5
201-XXX-XXXX	4
212-XXX-XXXX	3
908-XXX-XXXX	2
XXX-XXX-XXXX	1

shows how the data would be stored for the fixed-position addresses from Figure 1. The top address has the highest priority (7) and the bottom address has the lowest priority (1). Table 6 shows the implicit priorities for the variable-position addresses from Figure 2. With the ternary-CAM loaded in this ordered way, the CAM generates correct matches in one search cycle.

When a packet arrives at the switch, the system extracts its packetAddress that is used to *search* the CAM. If there are multiple matches, the CAM picks the one with the largest priority. For example, if the packetAddress of 11010000 is matched against Table 6 it results in three matches: XXXXXXXX (priority 1), 1XXXXX00 (priority 4), and 11X10X00 (priority 6). In this case, the switch correctly chooses the 11X10X00 entry. After initialization, the CAMs return the associated information or index in two cycles: *search-read* (CAM search and CAM read).

If an old entry must be removed from the table, the system selectively deletes that entry. But, in the general case, if a new entry needs to be loaded into the table, the

Table 7
Fixed-position Address Priority

Address (decimal)	Priority (binary)
201-829-4698	11
201-829-XXXX	10
201-876-XXXX	10
201-XXX-XXXX	01
212-XXX-XXXX	01
908-XXX-XXXX	01
XXX-XXX-XXXX	00

Table 8
Variable-position Address Priority

Address (ternary)	Priority (binary)
11110XX0	11
11X10X00	10
1X111X00	10
1XXXXX00	01
1XXXXXXX	01
1XXXXXX1	01
XXXXXXXX	00

system must totally re-initialize the CAM and re-insert all entries in the proper order. There are, however, a number of techniques (see below) that can improve the update performance for specific applications and CAMs.

Method T1 is fast, has low complexity, and works well for all types of hierarchical addresses: particularly variable-position hierarchical addresses. The main drawback is the slow updating.

6.2: T2 (Multiple-Cycle Single-Logical CAM)

This section describes how to simplify the update operation by replacing the inherent priority (used in method T1) with an explicit priority field added to each word. This priority is based on the address depth in the hierarchy; the deeper the address, the higher the priority.

The system loads the CAM words with the addresses, the associated information (or index), and adds a priority field proportional to its depth. Table 7 shows the priority filed associated with the fixed-position addresses from Figure 1. Table 8 shows the priority field associated with the variable-position addresses from Figure 2. Resolving the priority with the priority field takes multiple cycles. Note that the order in which entries are added is irrelevant.

When a packet arrives at the switch, the system extracts its packetAddress. This packetAddress is used to *search* the CAM. Each cycle the system combines the packetAddress with a different priority word. In the first cycle of the search, the system sets the priority field equal to all X's, except for the *most significant* bit which is set equal to 1. Only those address which match and have the most significant bit of the priority equal to 1 produce a match. If the packetAddress is 11101100, the address-priority word is 11101100-1X. The top three addresses could have produced a match based on the priority field; but all fail based on the packetAddress field.

All subsequent cycles depend on the result of the previous operation. There are three steps each cycle:

- If there was no match in the previous cycle, then the least significant 1 in the priority field is replaced with a 0,
- The most significant X, if one exists (which is true for all but the final cycle), is replaced by a 1,
- Search with the new address-priority word.

Based on this algorithm, the final two cycles in our example work as follows. Because there was no match in the first cycle, the system searches against 11101100-01 in the second cycle. Because there is match in the second cycle (against 1XXXXX00-01), the system searches against 11101100-01 in the final cycle. The final match guarantees giving a single correct word and the system reads the associated information or index (from 1XXXXX00).

The total number of cycles is two more (three more for indirect addressing) than the number of bits in the priority (which is the log of the depth of the hierarchy): *search-search-...-search-read* (CAM search, CAM search,...,CAM search, CAM read).

If a new entry needs to be loaded into the routing table, the system adds the entry into the next free location with the associated priority field for an address at its depth in the hierarchy. If an old entry needs to be removed from the table, the system selectively deletes that entry.

Method T2 is slower and more complex than method T1; but allows fast updates.

6.3: T3 (Single-Cycle Multiple-Logical CAM)

This section briefly considers a variation of method B3, using multiple logical ternary-CAMs, for application with variable-position hierarchical addresses. The technique is almost identical to method B3, except that the logical CAMs are split by priority (proportional to the depth in the hierarchy) rather than masks. In other words, the entries with priority 11 (from Table 7) would be put in one CAM, those with priority 10 in another, and so on. The prioritizing logic on the output side of the CAMs would filter out all

Table 9

Comparing the six CAM routing table lookup techniques

#	# Cycles	# Logical	Type	Best Suited Address Structure	Drawbacks
B1	Single	Single	Binary	Flat	-
B2	Multiple	Single	Binary	Fixed hierarchical	Latency, complexity
B3	Single	Multiple	Binary	Fixed hierarchical	Extra hardware
T1	Single	Single	Ternary	All hierarchical	Slow Updates
T2	Multiple	Single	Ternary	Variable hierarchical	Latency, complexity
T3	Single	Multiple	Ternary	Variable hierarchical	Extra hardware

CAM outputs except that with the highest priority.

Method T3 is fast, allows fast updates, and has low complexity; but requires prioritization logic.

7 Conclusion

This paper describes six CAM-based systems for general address translation. Table 9 summarizes the results. The choice depends mainly on the address format. Flat addresses can be searched in one cycle with method B1. For variable-position hierarchical CAMs, method T1 is preferred if slow updates are tolerable, else method T3 is preferred. For fixed-position hierarchical addresses, methods B3 or T1 are preferred.

Although software based address lookup is used successfully on a wide variety of address structures, these methods will not scale to large high speed networks. Applications that only need flat addressing and a small local address space may continue to use RAM-based table lookup; but, as the speed of communication switches increase and the size of the packet remains constant - or even decreases because of the jitter requirements of multimedia applications - the CAM seems increasingly essential for general address translation. Hashing is a possible alternative, but it is complex and its performance depends on traffic patterns. The size limitations of CAM is a potential drawback, but with cashing and flushing (see Appendix A), the limited size should be adequate - particularly with hierarchical addresses.

There are interesting areas for future research related to the use of multiple matches. One example of using multiple matches is if the best route is overloaded or broken, it is desirable to know alternative routes. Similarly, in multi-path routing alternative routes can be used to distribute the load. Although beyond the scope of this paper, it is relatively easy to adapt the various methods (B2, B3, T1, T2, and T3) to give more than the best route.

Acknowledgments

We thank Dan Wilson, who participated in the early discussions of the multiple logical CAM system, and Dave Sincoskie, who suggested looking into multiple access techniques. Furthermore, we thank Chase Cotton, Gary Hayward, and Mike Kramer for reviewing the paper and providing some useful suggestions.

References

[1] T.B. Pei, C. Zukowski, "VLSI Implementation of Routing Tables: Tries and CAMs", *Proceedings of IEEE Infocom '91*, Bal Harbour, Florida, April 1991.

[2] H. Yamada, Y. Murata, T. Maeda, R. Ikeda, K. Motohashi, & K. Takahashi, "Real-time String Search Engine LSI for 800-Mbit/sec LANs," *Proceeding of IEEE Custom Integrated Circuits Conference*, pp. 21.6.1-21.6.4, 1988.

[3] L. Chivin & R. Duckworth, "Content-addressable and associative memory: Alternatives to the ubiquitous RAM," *IEEE Computer Magazine*, pp. 51-64, July 1989.

[4] A. J. McAuley & C. J. Cotton, "A Reconfigurable Content Addressable Memory," *IEEE Journal of Solid State Circuits*, Vol-26, No.3, pp.257-2621, March 1991.

[5] J. Mogul, J. Postel, "Internet Standard Subnetting Procedure", *RFC-950*, USC/Information Sciences Institute, August, 1985.

[6] P. F. Tsuchiya, "Efficient and Flexible Hierarchical Address Assignment", Proceedings of ISOC INET '92, Kobe, Japan, June 1992.

[7] P. F. Tsuchiya, "On the Assignment of Subnet Numbers", *RFC-1219*, USC/Information Sciences Institute, April, 1991.

[8] D. E. Knuth, *The Art of Computer Programming, Vol. 3 (Sorting and Searching)*, Addison Wesley, 1973.

[9] J. P. Wade & C. G. Sodini, "Dynamic Cross-coupled Bitline Content Addressable Memory Cell For High Density Arrays," *IEEE Journal of Solid State Circuits*, Vol-22, No.2, pp.119-121, Feb. 1987.

Appendix A Managing the CAM

The cost of a CAM word is too large to allow wasting the address space with words that are no longer needed. CAMs have inherently much smaller capacity than an equivalent cost RAM. Not only do CAMs require many more transistors, they also are typically not on the cutting edge of technology. Therefore, we cannot expect Megabit CAMs soon. To overcome the size limitation, flushing and cashing can be applied.

When the table entries are short lived, garbage collection becomes important. It is possible that a connection close message might be lost or never generated, in which case we require that the entries that are not being used be flushed from the system. The flushing method used in the RCAM [4] ameliorates this garbage collection problem. A flush identifies those CAM words that have not been accessed (i.e. written, read, or searched for) since the last flush operation. CAM words thus identified are marked as "old." A second flush will mark any "old' locations which still have not been accessed as free for reuse. Thus the CAM automatically performs LRU database management with no system overhead - except deciding the frequency of the flush. Garbage collection or flushing is particularly important where the addresses are changing rapidly, such as in VCI translation for an ATM switch.

If table entries are longer lived, especially when addresses need to be around for a long time but are only used rarely, cashing is desirable. Cashing is a standard memory management technique that can equally well be applied to CAMs as to RAMs. The only difference is that the primary (fastest) storage medium is a CAM. When a CAM fills up, the least recently used (LRU) CAM locations can be backed up into RAM or other secondary storage. When the system has a miss it can fetch the missing entry from the backup storage.

Appendix B Fast Ternary-CAM Updates

Although updating need only be executed when routing table entries are added, which we assume is relatively infrequent compared to the frequency of lookups, some applications might consider exploiting the characteristics of a given ternary-CAM or to improve update efficiently.

For example, consider a FIFM CAM that allows the selective removal of all accumulated matched entries (these features exist on a new version of the binary RCAM [4]). With this CAM, any entry can be added by first doing an accumulated match on 1) the entry being added, and 2) all smaller masks than the entry being added. After these matched entries are removed, the new entry is added, followed by those that were removed, in order of largest mask first. This technique works efficiently because an entry only need be added before those entries above it in the addressing hierarchy.

With the RCAM, some care must be taken when removing entries, because the next add will fill the location that was previously removed. A simple remove algorithm is to follow any remove with an add of a dummy entry (one that has an invalid address and a full mask, and therefore won't accidently match a valid packet address). This algorithm will require occasional garbage collection to clean out all the dummy entries. It is easy to imagine more sophisticated remove algorithms, where the priorities of the dummy entries are remembered, allowing them to be selectively filled during the add operation.